DSA Homework 7: Graph Traversal and Components

David Tarazi

March 26, 2020

1 Topological Ordering Theory

In order for a topological ordering to occur, any following node connection must be greater than the current one. I think it would be best do use a modified depth first search to traverse the graph and compare the values of the nodes as we traverse. Meanwhile, we can check for cycles to avoid unnecessary checks and give more failure modes. This task can be achieved using the pseudocode below.

Algorithm 1 DFS Topological Ordering

```
1:  create 3 sets: unvisited, processing, processed                ▷ States for nodes
2:  add all nodes to unvisited
3:  while unvisited set is not empty do                            ▷ To get all starting nodes in graph
4:      select a starting node, u
5:      create a stack, S, and add u to S                          ▷ Stack for current loop in graph
6:      remove u from unvisited set
7:      while S is not empty do                                    ▷ Loops through connected nodes until processed
8:          pop u from S and add u to processing                   ▷ u is current node being processed
9:          if any neighbor is in the processing set then
10:             return False                                       ▷ This would mean a cycle, breaking order
11:         for each processed neighbor do
12:             if neighbor is less than u then
13:                 return False                                   ▷ Topological order broken
14:         for each unvisited neighbor do
15:             if neighbor is less than u then
16:                 return False                                   ▷ Topological order broken
17:             add to S and remove from unvisited
18:     move all nodes in processing set to processed set          ▷ Nodes from this component are processed
19: return True                                                    ▷ Once we have traversed the graph and passed all checks
```

The way this algorithm works is by 1: checking for cycles and 2: checking if the next neighbors are less than the current neighbor to identify breaking points in topological order. The way this is done is by tracking nodes in 3 categories: unvisited, processing, and processed. The unvisited nodes are ones that have not been checked and with those we check if they break the rule and if not add them to the stack of nodes to be processed. The processing category is necessary to keep track of which nodes we are looking at in the current section of the graph. If any of the neighbors during this same loop is already in the processing category, we know we have reached a cycle and broken the rule. Lastly, we must move these processing nodes to the processed category once we have finished the current component. With this mark, we know for further down the line, if we check a processed node, we don't have to continue looking at the other nodes connected to this one.

2 Practice

See dtarazi.py for source code. Discussion of problem 3 is below.

For the random binomial graph probability problem, I saw that the smallest probability required to create a single component graph dropped at an exponential decay rate (roughly). At first it didn't make sense to me, but I realized that with more nodes, each node now has another node that it can connect to to create a single component graph which increases by n every time a single node is added, which makes sense. The results are shown below.

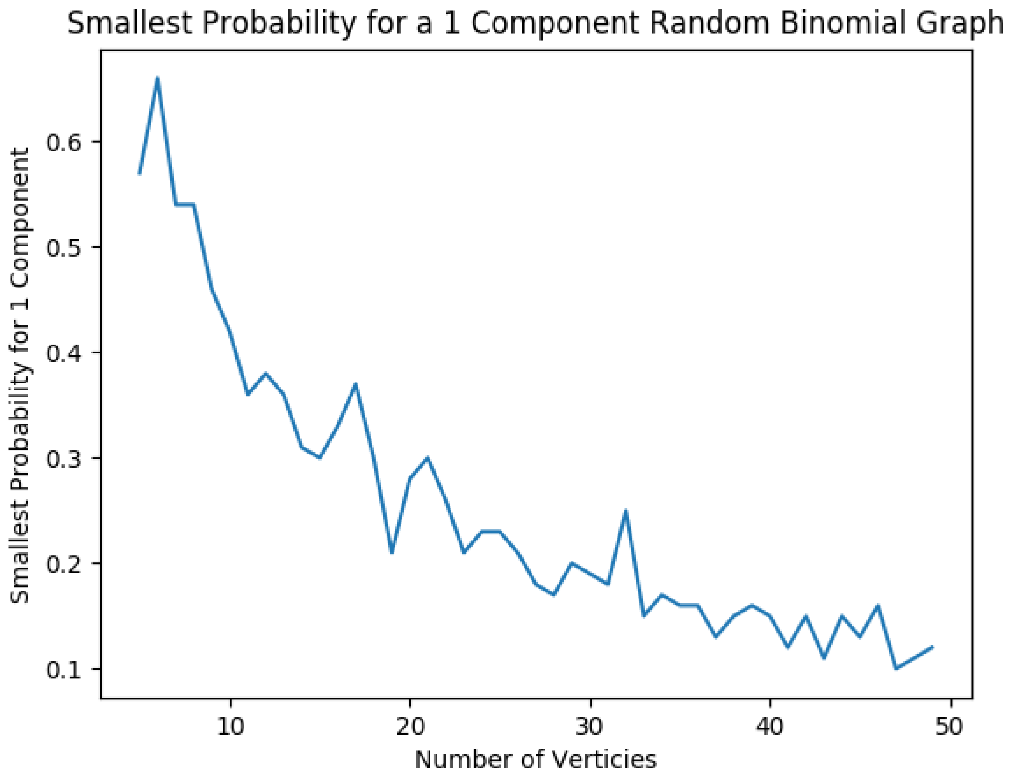

Figure 1: The number of nodes in a random binomial graph compared to the smallest probability for there to only be one component within the graph.